****

**Laboratório de Computadores**

**2018/2019**

Pen-Goo



Grupo T4G10:

Daniel Ferreira Brandão, up201705812, up201705812@fe.up.pt

Pedro Miguel Moás, up201705208, up201705208@fe.up.pt

Índice

[1.Instrucões de Utilização 3](#_Toc534576400)

[1.1. Ecrã Inicial 3](#_Toc534576401)

[1.2Menu Principal 3](#_Toc534576402)

[1.3 Draw 6](#_Toc534576403)

[1.4 Play (Pen-Goo) 8](#_Toc534576404)

[1.5 Draw Together 10](#_Toc534576405)

[1.6 Rainbow Snake 10](#_Toc534576406)

[1.7 Flappy Rainbow 11](#_Toc534576407)

[2. Estado do Projeto 12](#_Toc534576408)

[3. Organização / Estrutura de código 14](#_Toc534576409)

[Mouse 14](#_Toc534576410)

[Rtc 14](#_Toc534576411)

[Timer 14](#_Toc534576412)

[Keyboard 14](#_Toc534576413)

[Interrupts 14](#_Toc534576414)

[PenGoo 15](#_Toc534576415)

[Bitmap 15](#_Toc534576416)

[Bitmaps 15](#_Toc534576417)

[Sprite 15](#_Toc534576418)

[Layer 15](#_Toc534576419)

[Canvas 15](#_Toc534576420)

[Textbox 16](#_Toc534576421)

[Clock 16](#_Toc534576422)

[Emote 16](#_Toc534576423)

[Wordpicker 16](#_Toc534576424)

[Snake 16](#_Toc534576425)

[Flappy 17](#_Toc534576426)

[Game\_info 17](#_Toc534576427)

[Call Graph 19](#_Toc534576428)

[4.Detalhes de implementação 20](#_Toc534576429)

[5.Conclusão 20](#_Toc534576430)

[Avaliação da unidade curricular 21](#_Toc534576431)

# 1.Instrucões de Utilização

## 1.1. Ecrã Inicial

Ao iniciar o jogo, é mostrado o ecrã inicial. Neste é possível ver as horas / data atual e avançar para o menu principal, pressionando a tecla “Enter”.

Figura 1- Ecrã inicial

## 1.2Menu Principal

A partir do menu principal, é possível aceder a todos os modos de jogo e respetivas instruções, assim como o botão de saída.

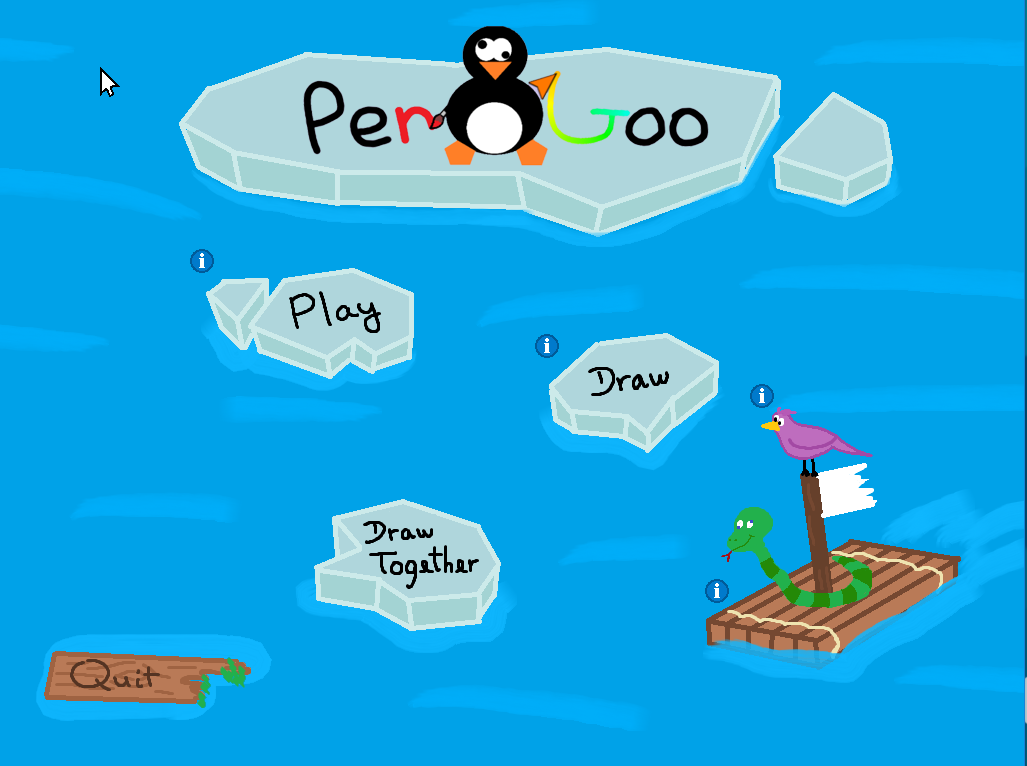


Figura 2 - Menu Principal

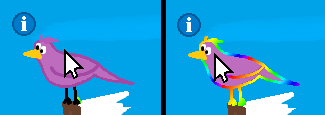
Para navegar neste menu, é utilizado o rato. Passar com o rato por cima de um dos botões faz com que esse fique “highlighted” , e clicar com o lado esquerdo do rato seleciona a opção correspondente. //acho que falta falar aqui das colisões do rato com os botões

Figura 3- "hovering" com o rato

Os modos de jogo disponíveis são: Play, Draw , Draw Together , Rainbow Snake e Flappy Rainbow.



Figura 4- Instruções do modo Flappy Rainbow

## 1.3 Draw

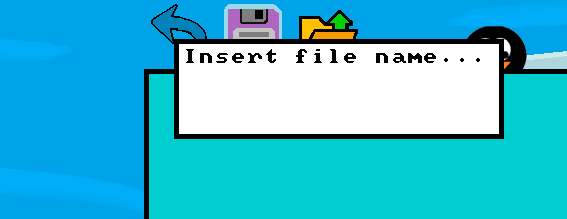
Modo singleplayer, onde o utilizador pode criar os seus próprios desenhos.

Figura 6 - Inserir nome para guardar ficheiro

Figura 5 - Modo Draw

Para além de começar um desenho em branco, é possível fazer load a ficheiros em formato bitmap e guardar as criações.

Para criar o desenho, o utilizador tem acesso a múltiplas ferramentas de desenho:

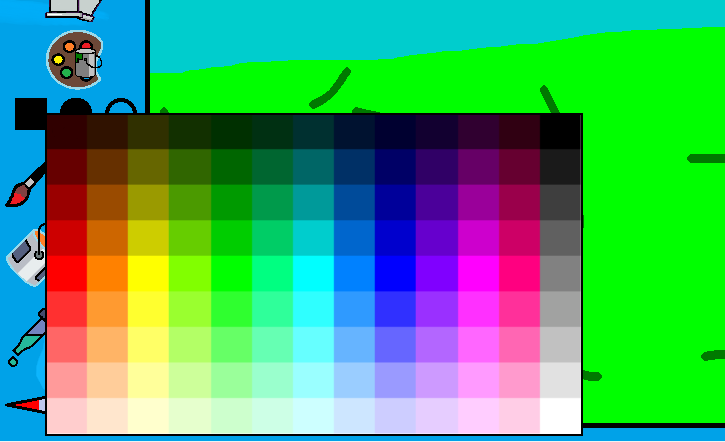
* ***Slider* de grossura** – arrastar o cursor para aumentar ou diminuir a grossura do pincel.

Figura 7 - Palete de cores

* **Palete de cores –** abre uma palete de cores, de onde se pode escolher a cor principal (Butão esquerdo do rato) e secundária (Butão direito do rato)
* **Pincel –** ferramenta de desenho principal**.** Cor principal (botão esquerdo do rato) e secundária (botão direito do rato)
* **Arco-Íris –** Pincel com gradiente arco-íris
* **Balde –** Preenche uma área com a mesma cor
* **Borracha –** Usar o rato para apagar. Botão direito utiliza a cor principal e botão esquerdo a secundária.
* ***Color-Picker* –** Clicando em qualquer área do ecrã consegue-se obter qualquer cor.
* **Lixo –** Apaga o desenho.
* **Formas –** Permite desenhar formas com dois cliques do rato. Retângulo: Dois cliques representam dois cantos opostos. Círculo/Circunferência: Primeiro clique para o centro, segundo para o raio.

Para a maioria das ferramentas é possível utilizar Cor principal (botão esquerdo do rato) e secundária (botão direito do rato), exceto para a borracha, na qual acontece o inverso.

Para além das ferramentas de desenho é possível:

* Ver cor principal e secundária atual utilizando os guaches de cor
* Anular a última ação utilizando o botão de UNDO ou pressionar “Ctrl” + “Z”

Para selecionar maioria destas ferramentas, utilizam-se botões clicáveis. A palete de cores abre um pop-up, onde se pode selecionar a cor desejada e o slider de grossura é um *slider* interativo.

Ao selecionar uma ferramenta, o cursor muda de imagem para a ferramenta escolhida e ao mudar de cor principal, a cor deste também muda.

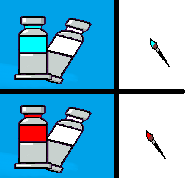


Figura 9-Mudança da cor do cursor

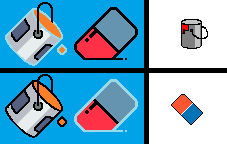


Figura 8- Mudança da forma do cursor

## 1.4 Play (Pen-Goo)

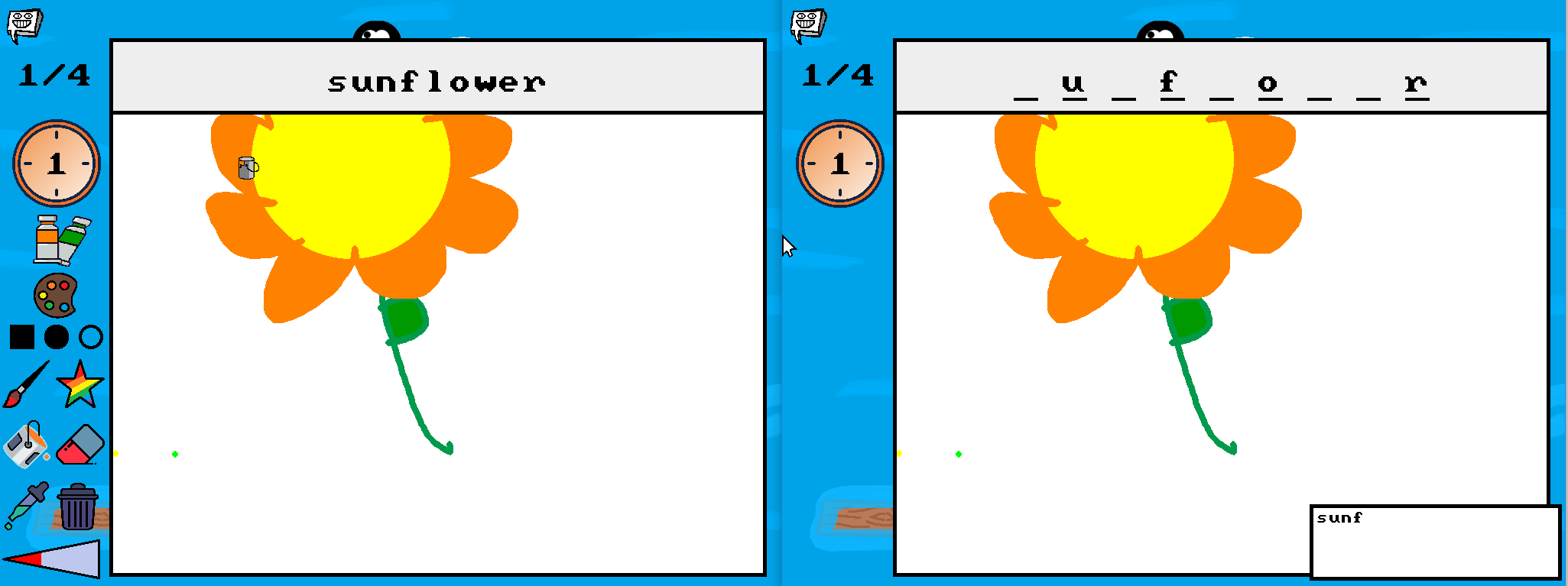
Modo principal do projeto, trata-se de um jogo de adivinhar cooperativo para dois jogadores. O objetivo do jogo é adivinhar a palavra que o parceiro está a desenhar o mais rápido possível. Cada sessão tem 4 rondas. O papel de cada jogador é escolhido quando se começa a jogar aleatoriamente. Cada uma desenvolve-se da seguinte forma: É dado ao jogador que vai desenhar uma palavra aleatória do dicionário, o jogador tenta representar o que desenha, e a mesmo tempo, noutro computador, utilizando a porta de série, o outro jogador tenta adivinhar o que o parceiro está a tentar representar , usando o teclado, o mais rapidamente possível. O desenho é transmitido em tempo real para o computador do parceiro. O jogador que desenha tem acesso a todas as ferramentas de desenho do modo “Draw”.

Figura 10 - Pen-Goo (vista dos 2 jogadores)

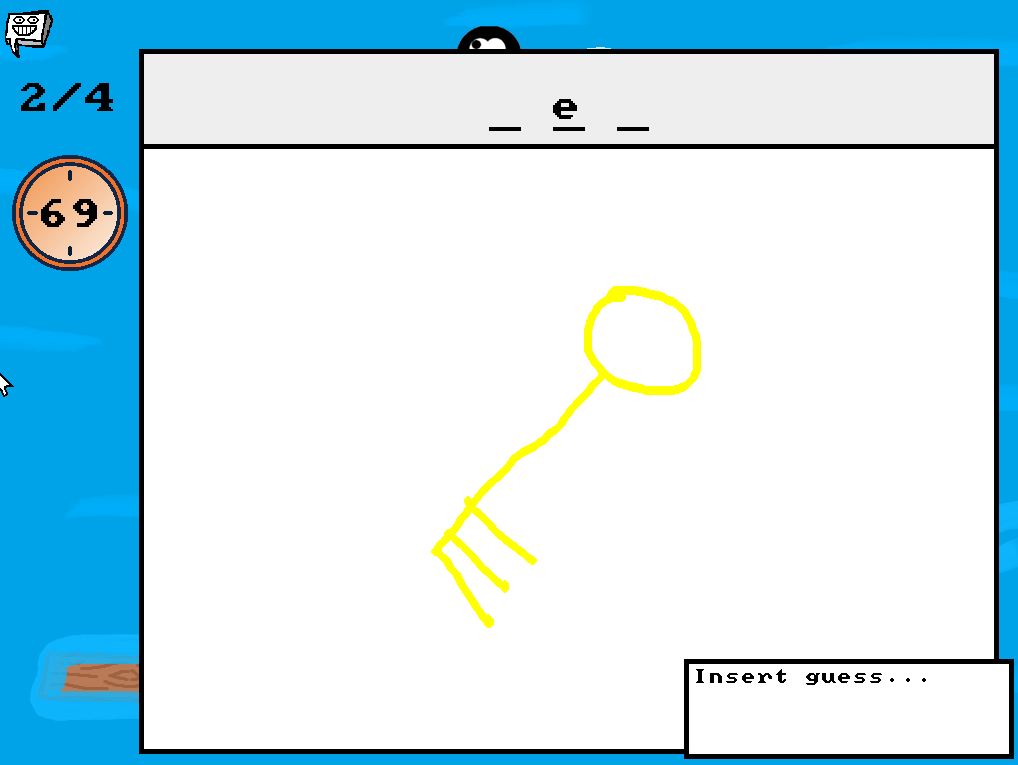


Figura 11 - Caixa de texto para tentar adivinhar a palavra

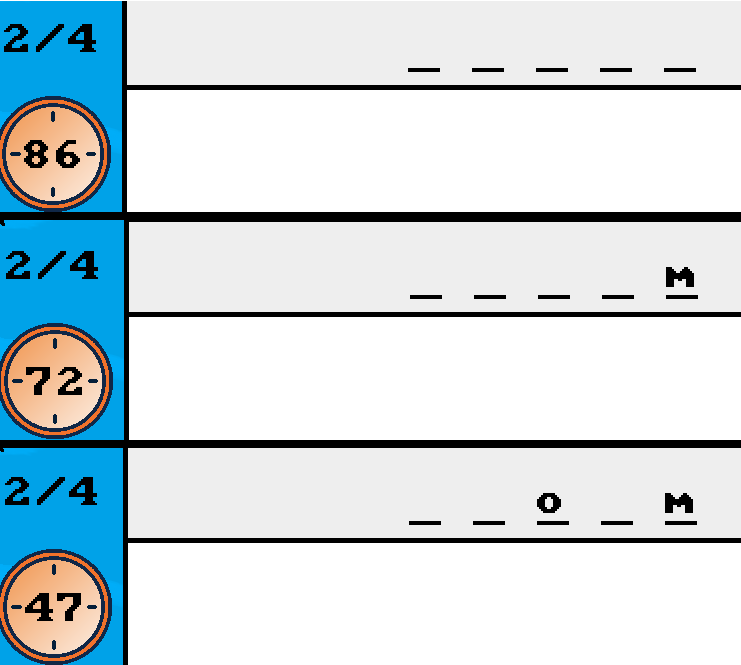
Para além do desenho do seu colega, o jogador que tenta adivinhar também saberá quantas letras tem a palavra que tenta adivinhar, e com o passar do tempo algumas delas vão sendo reveladas.

Figura 13 - Ecrã de palavra adivinhada com sucesso

Figura 12 - Letras a serem reveladas à medida que o tempo passa

Quanto mais rápido o jogador adivinhar, mais pontos o grupo ganha na ronda.

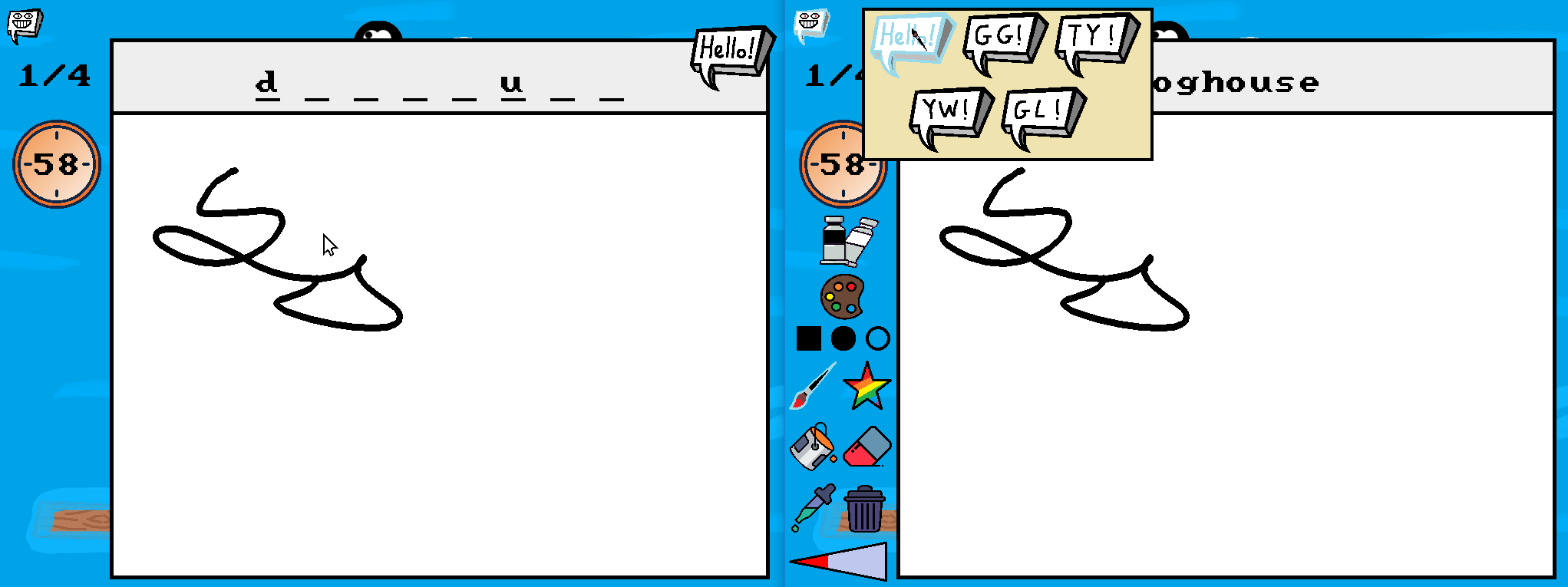
Os jogadores podem comunicar entre si utilizando emotes.

Figura 14 - Sistema de emotes (visto da perspetiva dos dois jogadores)

## 1.5 Draw Together

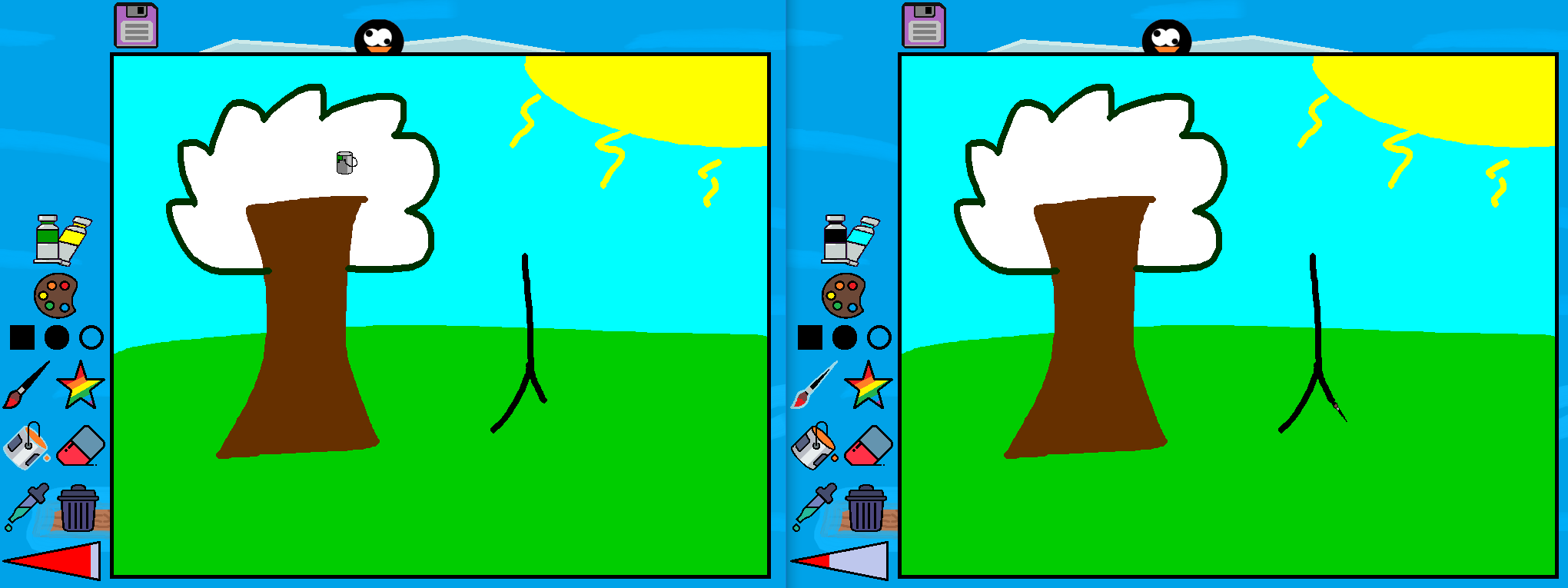
Modo de jogo igual ao “*Draw*”, mas com um parceiro, através da porta de série. Os desenhos estão sincronizados em tempo real. Todas as ferramentas de desenho estão disponiveis para ambos os utilizadores, e ainda têm a possibilidade de guardar o desenho, tal como no modo “Draw”.

Figura 15 - Modo Draw Together

## 1.6 Rainbow Snake

Figura 17-Ecrã de resultados do rainbow snake

Figura 16- Rainbow Snake

O modo de jogo *rainbow snake* trata-se de um modo de jogo secundário, que se diferencia bastante do resto do projeto. O objetivo é sobreviver o maior tempo possível, controlando a *rainbow snake*. Com o passar do tempo a *snake* vai perdendo vida e ganhando velocidade, por isso é necessário utilizar os *power-ups* disponíveis para sobreviver e maximizar a pontuação. A snake é controlada com as setas do teclado, podendo andar em qualquer direção. Atingir os obstáculos faz com que a snake perca vida e atingir uma parede faz com que seja refletida e que perca velocidade.



Figura 18 – Power-ups de rainbow snake, visualizados nas instruções do jogo

O highscore é gravado num ficheiro de texto.

## 1.7 Flappy Rainbow

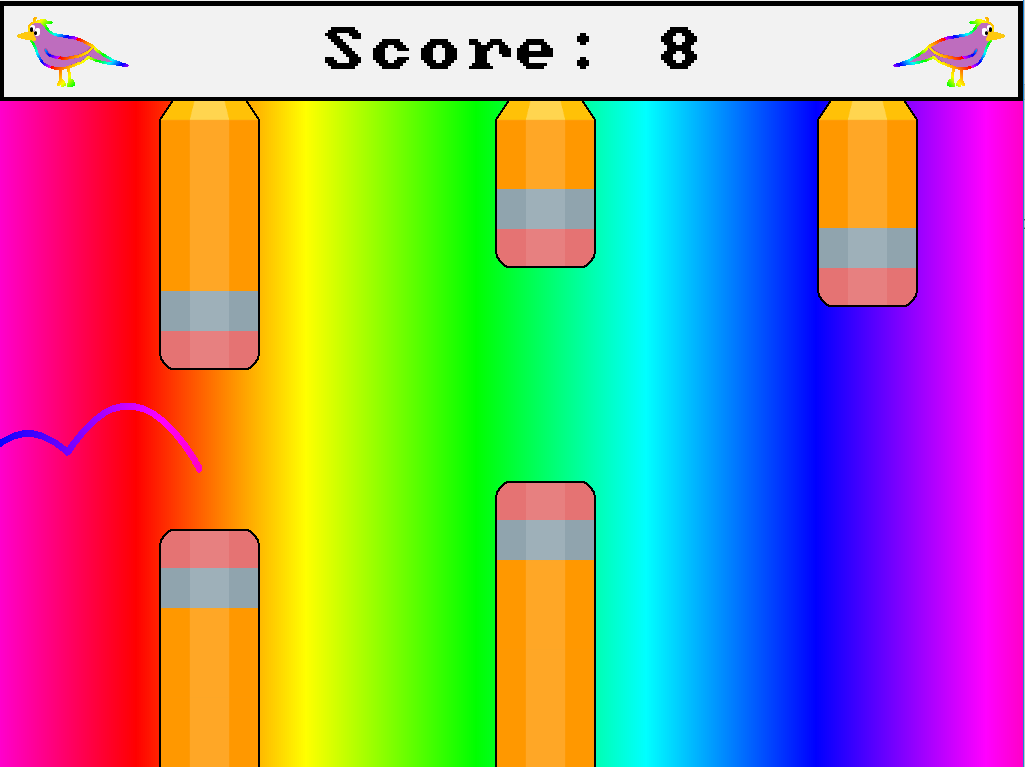
Outro modo de jogo secundário, *Flappy Rainbow*, baseado no conhecido jogo mobile, *Flappy Bird*, consiste em sobreviver o máximo de tempo possível, evitando obstáculos, apenas com salto (tecla “space bar”) e conhecimento da física do jogo. O *flappy rainbow* tem velocidade e aceleração (para baixo), para tornar o jogo mais consistente com a realidade. O jogador perde quando o *flappy rainbow* colidir com um dos obstáculos. Tal como o *rainbow snake*, o highscore é gravado num ficheiro de texto, para registo futuro.

Figura 19 - Flappy Rainbow

# 2. Estado do Projeto

|  |  |  |
| --- | --- | --- |
| Dispositivo | Utilização | Interrupção? |
| Timer | Controlo da frame rate nos minijogos, revelar letras periodicamente, animações a mostrar scores. | Sim |
| Teclado | Escrever texto no ecrã, controlar *rainbow snake* e *flappy rainbow*, alguns *shortcuts.* | Sim |
| Rato | Navegação nos menus, desenhar, escolha de ferramentas de desenho. | Sim |
| Placa Gráfica | Visualização de imagens, desenho e menus. | Não |
| RTC | Mostrar data e hora, atualização do relógio de jogo. | Sim |
| Porta de série | Modos de jogo multiplayer “Play” e “Draw Together” | Sim |

* **Timer**: O timer 0 é usado para controlar a framerate, atualizar o desenvolvimento do jogo nos modos “*Rainbow Snake*” (*snake.c*) e “*Flappy Rainbow*” (*flappy.c*), para controlar o tempo entre letras reveladas (*wordpicker.c – reveal\_letter()*), e para mostrar a pontuação final não espontaneamente.

Está implementado em**:** timer.c

* **Teclado:** O teclado é utilizado para escrever no ecrã , a partir de caixas de texto (*textbox.c*) (adivinhar palavra, guardar nome de ficheiro, fazer load a ficheiro) , para sair de certos modos de jogo e fazer *undo* no desenho (“Ctrl” + ”Z”) , mas também para controlar a *snake* (setas) (*snake.c – snake\_turn\_left/right()*) e o flappy (“Space bar”) (*flappy.c -flappy\_jump()*).

Está implementado em: keyboard.c

//acho que não e isto que eles querem com os ficheiro / funções.... problema deles

* **Rato:** Usado para controlar o cursor, que é utilizado para navegar nos menus e selecionar ferramentas, a partir de botões e desenhar, não só para escolher onde desenhar, mas também para escolher se se utiliza a cor principal (botão esquerdo do rato) ou secundária (botão direito do rato).

Está implementado em: mouse.c

* **Placa Gráfica:** Este dispositivo é utilizado para mostrar todas as imagens usadas no programa (bitmap, RGB(8;8;8)). O modo de vídeo utilizado é 118 (1,024×768 (24 bits p pixel)). (*proj.c*)

Double buffering está implementado no código do programa, mas não é utilizado, pois devido ao facto de que se trata de uma ferramenta de desenho, não muito focada em atualização frequente do ecrã inteiro, não faria muita diferença, tornando a experiência mais lenta até. (*video.h*)

Foi utilizada colisão de *sprites* para o cursor e botões, e hitboxes para os modos de jogo “Rainbow Snake” e “Flappy Rainbow”. (*sprite.c - is\_on\_button()*) (*sprite.c- check\_hitbox\_collision()*)Um sistema de *layers* está a ser utilizado de modo a ser possível desenhar facilmente janelas por cima de outras sem perder a informação que está “por baixo”. (*layer.c*)

A fonte utilizada para a textbox, para mostrar as pontuações dos jogadores, tempo do relógio, etc., é retirada de um bitmap que contém todos os carateres, que depois são retiradas individualmente a partir de um algoritmo. (*textbox.c*)

Está implementado em: video.c vbe.c

* **RTC :** O Real Time Clock utiliza alarmes de 1 segundo para atualizar a hora / data atual periodicamente no Ecrã inicial (*clock.c – update\_big\_ben()*) e para atualizar o relógio de jogo (90s por ronda no máximo). (*wordpicker.c – wordgame\_tick\_clock()*)

Está implementado em: rtc.c

* **Porta de série:** A porta de série utiliza FIFO, Interrupções e, por vezes, *polling*, para transferir informação entre dois computadores diferentes. A porta de série é um dispositivo crucial para o funcionamento dos modos “Play” (Adivinhar) e “Draw Together”, nos quais se transferirão até cerca de 1.5 KBytes por segundo, potencialmente ao mesmo tempo, de um computador para outro. Através desta são enviadas várias informações, como quem será o primeiro a desenhar, que um jogador está pronto, qual a palavra a adivinhar, se o jogador adivinhou a palavra, que o outro computador desenhou uma linha, numa certa posição, com uma certa cor, entre outras. Para facilitar a gestão da informação recebida/enviada, para ambos os computadores saberem o significado de cada byte recebido, e para não haver dessincronizações que prejudiquem muito o desenvolvimento do jogo, criou-se um protocolo relativamente simples, para a comunicação neste projeto, ilustrado na imagem seguinte. As mensagens são então constituídas por um prefixo, que contém também o tipo de mensagem e o seu tamanho, e por um sufixo. Mesmo que haja informação que por vezes seja redundante, considerou-se que por segurança guardar-se-ia na mesma.

Implementado em uart.c, uart\_protocol.c, e uart\_wordgame.c (este último para aplicar o protocolo a casos do jogo em concreto)

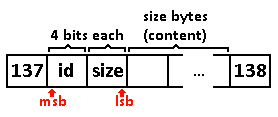
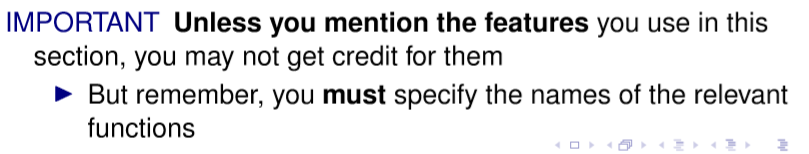


Figura 20 – Protocolo de comunicação usado para o UART

* **Porta de série:** Mt importante,...



# 3. Organização / Estrutura de código

### Mouse

Código importado do lab4. Utiliza-se para subscrever as interrupções do rato e interrupt handler do rato.

### Rtc

Lê a informação dos registos do real-time-clock.

Subscreve alarmes periódicos de 1 segundo, para ler a data/horas em todos os segundos. Uso das struct Date / Mini\_Date para obter as datas a partir das funções rtc\_get\_date/mini\_date();

### Timer

Código importado do lab2. Utiliza-se para subscrever as interrupções e interrupt handler do timer.

### Keyboard

Código importado do lab3. Utiliza-se para subscrever as interrupções e interrupt handler do keyboard.

Vbe

Video

### Uart

Utiliza-se para subscrever as interrupções e interrupt handler do UART, mas também outras funções, como ler o seu status register, ler ou enviar um byte através da porta de série.

### Uart\_protocol

Código utilizado para definir, criar e enviar mensagens pela porta de série, através de um protocolo definido (descrito na secção anterior).

### Uart\_wordgame

Utiliza-se para enviar mensagens e processar mensagens recebidas, no contexto do projeto, por exemplo, permite enviar uma mensagem indicando que deve ser desenhada uma linha desde uma posição até outra, com uma certa cor e espessura, como também processar essa mesma mensagem.

### Interrupts

Módulo onde existem funções para dar subscribe e unsubscribe e receber notificações de todos os periféricos utilizados de forma mais limpa e intuitiva. Uso da struct Notification para, a partir da função GetNotification() obter todas as informações convenientes da notificação recebida.

### PenGoo

Este modulo é a base do projeto, contem a logica dos menus do projetor, de conectar jogadores em modos multiplayer – select\_drawer(), wait\_for\_collab(), wait\_for\_drawer() , wait\_for\_guesser(), de mudar as ferramentas / grossura e cor que está a ser utilizada - change\_tool().

//NAO SEI BEM O QUE ESCREVER MAIS

### Bitmap

Este ficheiro é utilizado para fazer load / save a ficheiros em formato bitmap, mas também para desenhá-los. Ao serem desenhados, os bitmaps podem ter transparência – representado pela cor ROSA (rgb (255,0,255)) e uma cor mudável – representado pela cor VERDE (rgb(0,255,0)). Esta permite mudar uma certa parte do bitmap, para qualquer cor desejada (Utilizado para mudar a cor do cursor para a cor primaria). Neste modulo estão também implementadas funções que rodam e recortam bitmaps - draw\_bitmap\_rotate() e ainda uma função que desenha o bitmap translucido - draw\_bitmap\_transp()

### Bitmaps

Modulo onde todos os bitmaps utilizados no projeto são lidos.

### Sprite

Módulo onde esta implementado as Sprites e todas as suas “Classes derivadas”. A sprite pode ser desenhada (tendo em atenção a transparecia e a sua cor mudável) , apagada e movida. A struct Cursor e as suas funções para fazer update , mudar de bitmap e cor (para poder representar a ferramenta e cor primária que estão a ser utilizadas) , a struct Hitbox (utilizada para o jogo da snake e Flappy) e as suas funções para mover e para detetar colisões , a struct Button (usada para navegação nos menus e escolha de ferramentas) e as funções para detetar se o cursor esta a fazer hover , fazer highlight ao botão e pressiona-lo , a struct iddle sprite , que se trata de uma sprite não interativo que pode mudar de cor (utilizado para mostrar cor primaria e secundaria atual) e a struct slider (utilizado para o slider dinâmico de grossura) e as suas funções para mover e atualizar estão implementadas neste ficheiro.

### Layer

Modulo onde esta implementado sistema de layers que permite desenhar sprites numa layer- draw\_on\_layer() em cima de outra layer , sem perder informação sobre o que esta por baixo. Graças a este sistema, é possível abrir janelas a utilizar o programa , a partir de funções que verificam qual é a layer que está em cima - Is\_top\_layer() e se certa layer é a do topo - is\_top\_layer().

### Canvas

Implementação da struct canvas que é utilizada como espaço de desenho para o utilizador. O canvas tem uma layer e limites. É aqui que se encontram as funções de desenho principais como canvas\_draw\_line(), canvas\_draw\_circle() , etc. . As ferramentas de desenho como o rainbow e bucket tool e as shapes estão aqui definidos.

O sistema de undo é feito utilizando as funcões canvas\_save\_drawing() e canvas\_undo().

### Textbox

A textbox é utilizada para escrever texto, quer pelo utilizador, ou para mostrar informações. É implementada utilizando uma struct que contem o bitmap da imagem da textbox, layer, tamanho ta fonte, coordenadas da textbox e do cursor utilizado para escrever texto.

A fonte utilizada é lida a partir de um ficheiro bitmap com a função loadLetterMap().

Para escrever numa caixa de texto é utilizada são utilizadas a funções textbox\_put() e textbox\_write(), e textbox\_backspace() para apagar.

Este módulo também contém duas funções que não utilizam a struct Textbox, draw\_char() e draw\_word(), mas são relevantes para o módulo pois permitem, respetivamente, desenhar no ecrã um carater e uma string, numa posição especificada.

### Clock

Neste ficheiro está implementado o relógio utilizado para mostrar as horas/data no ecrã inicial, que utiliza o rtc.

### Emote

Implementação do sistema de emotes que são enviados entre jogadores nos modos multiplayer a partir da serial port. Permite desenhar emotes no ecrã. Também tem funções para abrir ou fechar a “emote wheel”, através da função toggle\_emote\_wheel().

### Wordpicker

Este módulo trata de ler as palavras que vão ser adivinhadas / desenhadas de um ficheiro de texto (dicionário). Também contem o mecanismo de mostrar a palavra escondida que está a ser adivinhada utilizando as funções wordgame\_draw\_hidden\_word() para mostrar a palavra escondida (só com os ‘\_’ em vez de letras) e reveal\_letter() para revelar letras à medida que o tempo passa. Este módulo também contém a função verify\_guess() que verifica se a tentativa de adivinhar estava correta, ignorando diferença entre letras maiúsculas/minúsculas.

### Snake

Módulo que contém todo o modo de jogo secundário rainbow snake. As structs principais utilizadas neste modo de jogo são:

* Snake – contém as coordenadas, velocidade (e velocidade máxima), ângulo, vida, estado de invencibilidade, etc. Para controlar a snake as funções snake\_move() e snake\_turn\_right/left() são essenciais.
* Powerup – contém uma hitbox, bitmap , tipo de power , etc.
* Obstacle - contém uma hitbox, coordenadas e bitmap
* SnakeArena – contém os limites da arena.

A função snake\_move() verifica colisões com obstáculos , powerups e parede. Ao apanhar um powerup ou houver colisão com obstáculos a barra de vida da snake vai ser atualizada snake\_update\_hp\_bar().

No inicio do jogo diferentes obstáculos são colocados na arena de forma aleatório, mas a evitar overlapping - arena\_add\_obstacle(). Durante o jogo são adicionados á arena power ups - arena\_add\_powerup()

No final do jogo é mostrado os stats aos jogadores - snake\_showstats() que utiliza um bitmap translucido. O highscore é guardado num ficheiro de texto - saveSnakeHighscore() , loadSnakeHighscore().

### Flappy

Modulo onde esta implementado o modo de jogo Flappy Rainbow. As structs utilizadas são:

* Bird – contém coordenadas , velocidade , aceleração , hitbox , raio do círculo , etc. do pássaro
* FlappyTube – contém bitmap, hitbox e um booleano que indica se o obstáculo já foi passado.

Para o jogo avançar, é utilizada a função flappy\_tick() que avança o ecrã para a esquerda, utiliza a função flappy\_move() que atualiza a posição do pássaro com base na sua aceleração e velocidade e a função flappy\_collided() verifica se houve colisões com os obstáculos.

Para controlar o pássaro, a função flappy\_jump() é utilizada para fazer com que ele salte.

Os obstáculos são adicionados com a função flappy\_add\_tube().

No final do jogo é mostrado os stats aos jogadores - flappy\_showstats(). O highscore é guardado num ficheiro de texto - saveFlappyHighscore(), loadFlappyHighscore().

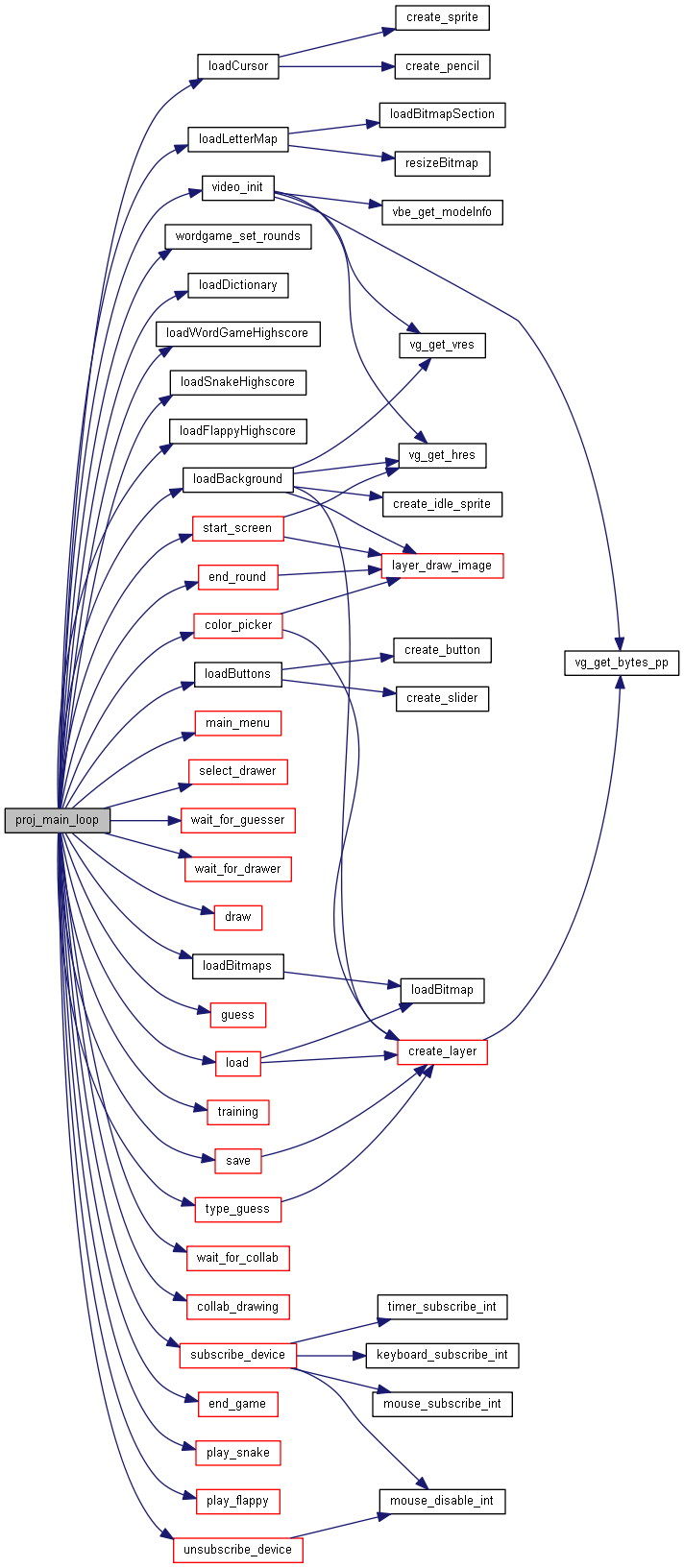
### Game\_info

Módulo que contem as instruções de cada modo de jogo, que estão disponíveis no menu principal.

Game\_State

|  |  |  |
| --- | --- | --- |
| Modulo | Peso RELATIVO (%) | Feito por |
| Mouse |  | Daniel/Pedro |
| RTC |  | Daniel |
| Timer |  | Daniel/Pedro |
| Keyboard |  | Daniel/Pedro |
| VBE |  | Daniel/Pedro |
| Video |  | Daniel/Pedro |
| Uart |  | Pedro |
| Uart\_Protocol |  |  |
| Uart\_Wordgame |  |  |
| Interrupts |  |  |
| PenGoo |  |  |
| Bitmap |  |  |
| Bitmaps |  |  |
| Canvas |  |  |
| Sprite |  |  |
| Textbox |  |  |
| Layer |  |  |
| Canvas |  |  |
| Clock |  |  |
| Emote |  |  |
| Wordpicker |  |  |
| Snake |  |  |
| Flappy |  |  |
| Game\_info |  |  |
| Game\_state |  |  |

## Call Graph



# 4.Detalhes de implementação

De um modo geral, achamos que maioria de o que foi utilizado no projeto foi bem abordado nas aulas laboratoriais de LCOM. No entanto para alguns dos módulos que deram mais trabalho, pois apenas foram falados nas aulas teóricas:

* **RTC** – apesar de não diferir muito dos outros dispositivos dados nas aulas, não nos foi óbvio qual era a maneira mais eficaz de atualizar o relógio. Acabamos por definir alarmes de segundo a segundo para saber quando atualizar o tempo.
* **Serial Port** – Embora não consideremos que seja o dispositivo mais difícil por uma grande margem, é certamente o mais problemático para dar Debug, por várias razões: 1. Dificuldade em analisar o trace.txt / output.txt pois não indicam qual dos computadores originou as mensagens, 2. tanto a transmissão como a receção podem estar a causar erros, 3. na maioria dos erros que foram ocorrendo, apenas um *poweroff* permitia a UART regressar a um estado funcional, entre outras.
* **Bitmaps** – Sendo completamente necessário para o projeto,
* **Layers** – Desde início do projeto achamos que era necessário implementar um sistema de layers, de modo a ser possível termos janelas “pop-up” para podermos escrever texto, escolher cores de forma menos limitada, mostrar scores , etc.
* **Colisões e hitboxes** – Para detetar se o cursor estava a fazer hovering com os botões, foi necessário aplicar um método de colisões com as sprites simples. Para além disso foram aplicadas hurtboxes nos modos de jogo secundário “Rainbow Snake” e “Flappy Rainbow” para detetar colisões.

Além dos módulos, algumas funções mais complicadas foram necessárias:

* **Algoritmos de círculos , linhas** – Para poder desenhar círculos e linhas não horizontais/verticais tivemos de utilizar algoritmos pré-existentes (<https://en.wikipedia.org/wiki/Midpoint_circle_algorithm> , <https://en.wikipedia.org/wiki/Bresenhams_line_algorithm> )
* **Balde** - A ferramenta de desenho balde foi alvo de muitas alterações ao longo do projeto. Para encontrar o algoritmo mais otimizado experimentamos várias versões (<https://en.wikipedia.org/wiki/Flood_fill>) até encontramos a mais fluida. Inicialmente pensámos num algoritmo recursivo que verificava os 4 píxeis adjacentes, chamando a função recursivamente por cada pixel que tivesse a mesma cor inicial. O algoritmo foi simples e direto, mas (não muito surpreendentemente) causava *stack overflow* para regiões muito grandes. De seguida tentámos um algoritmo que teria um efeito semelhante ao anterior, mas de forma iterativa, utilizando uma queue, que conteria os pixéis a tratar, sendo que por cada pixel da queue os 4 adjacentes seriam verificados e os que teriam a sua cor igual à inicial seriam colocados na queue. O algoritmo era bastante mais complexo, mas mesmo com um enorme conjunto de otimizações à implementação do algoritmo e da queue (havendo até uma tentativa de implementar a queue através de um array com capacidade de 1500000, sendo o máximo de eficiência **temporal** que atingimos), demorava quase 0.25s para encher o ecrã inteiro (embora pareça curto, estraga bastante a experiência do jogador). No final, tentámos um terceiro algoritmo, que analisa os pixéis numa linha horizontal, e para cada linha analisa as suas duas adjacentes recursivamente, o que refletiu-se ser quase instantâneo em qualquer situação.
* ...?

# 5.Conclusão

## Avaliação da unidade curricular

Na nossa opinião, LCOM foi uma unidade curricular que se começou de forma brusca devido a introdução repentina de vários mecanismos de trabalho simultaneamente. A introdução do SVN, Minix, programação em C e matéria de LCOM tudo ao mesmo tempo foi um choque desencorajador. No entanto, à medida que os labs passavam, a matéria ficava mais esclarecida, e a unidade curricular foi-se tornando ~~exponencialmente~~ mais interessante.

Devido ao certo cuidado e trabalho necessário para fazer os labs, LCOM tornou-se por uma grande margem a unidade curricular que tivemos de investir mais horas semanais, o que não se reflete de todo nos ECTS definidos para a disciplina.

Outro ponto a destacar é o atraso na avaliação dos labs e do miniteste, tendo em conta que o primeiro lab (lab2) foi entregue no dia 14 de Outubro, e até à entrega do projeto não houve uma única avaliação entregue, o que prejudica os alunos, pois não só não conseguem ter uma ideia clara do seu desempenho até ao momento, como também haverão certos erros nos labs que serão repetidos nos labs seguintes, o que poderia ser evitado.

Por outro lado, LCOM ajudou-nos a ser mais autónomos na aprendizagem de certos assuntos, e o aumentou o nosso conhecimento sobre a linguagem C. O ponto alto da unidade curricular foi sem dúvida este projeto.